



Red Hat Training and Certification

Student Workbook

Kubernetes 1.22 DO100A

Foundations of Red Hat Cloud-native Development

Edition 2



Red Hat
Learning Community

Join a community dedicated to learning open source

The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.



Network with tens of thousands of community members



Engage in thousands of active conversations and posts



Join and interact with hundreds of certified training instructors



Unlock badges as you participate and accomplish new goals



This knowledge-sharing platform creates a space where learners can connect, ask questions, and collaborate with other open source practitioners.

Access free Red Hat training videos

Discover the latest Red Hat Training and Certification news

Connect with your instructor – and your classmates – before, after, and during your training course.

Join peers as you explore Red Hat products

Join the conversation learn.redhat.com



Copyright © 2020 Red Hat, Inc. Red Hat, Red Hat Enterprise Linux, the Red Hat logo, and Ansible are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

A long-exposure photograph of a city street at night, showing light trails from cars and buildings in the background. A semi-transparent white box is overlaid on the upper left, containing the title text. A faint grid pattern is visible in the bottom right corner.

Foundations of Red Hat Cloud-native Development

Kubernetes 1.22 DO100A

Foundations of Red Hat Cloud-native Development

Edition 2 r00000000

Publication date 20220000

Authors: Enol Alvarez de Prado, Manuel Aude Morales,
Guy Bianco IV, Marek Czernek, Natalie Lind, Michael Phillips,
Eduardo Ramirez Ronco, Alejandro Serna Borja, Jordi Sola Alaball
Course Architects: Zachary Gutterman, Fernando Lozano
DevOps Engineer: Richard Allred
Editor: Sam Ffrench

Copyright © 2022 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are Copyright © 2022 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle American, Inc. and/or its affiliates.

XFS® is a registered trademark of Hewlett Packard Enterprise Development LP or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is a trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Document Conventions	ix
.....	ix
Introduction	xi
Foundations of Red Hat Cloud-native Development	xi
Orientation to the Classroom Environment	xii
1. Introducing Containers and Kubernetes	1
Introducing Containers and Container Engines	2
Introducing Kubernetes and Container Orchestration	5
Summary	7
2. Running Containerized Applications	9
Contrasting Kubernetes Distributions	10
Guided Exercise: Contrasting Kubernetes Distributions	14
Introducing Kubectl	27
Guided Exercise: Connecting Kubectl To Your Cluster	32
Running and Interacting with Your First Application	39
Guided Exercise: Running and Interacting with Your First Application	42
Summary	46

Document Conventions

This section describes various conventions and practices used throughout all Red Hat Training courses.

Admonitions

Red Hat Training courses use the following admonitions:



References

These describe where to find external documentation relevant to a subject.



Note

These are tips, shortcuts, or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on something that makes your life easier.



Important

These provide details of information that is easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring these admonitions will not cause data loss, but may cause irritation and frustration.



Warning

These should not be ignored. Ignoring these admonitions will most likely cause data loss.

Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services covered in Red Hat Training courses. Red Hat appreciates your patience during this process.

Introduction

Foundations of Red Hat Cloud-native Development

Foundations of Red Hat Cloud-native Development (DO100a) is designed for IT professionals without previous cloud application deployment experience to learn basic Kubernetes skills. This course is a part of a three-course specialization. In this specialization, you will run, deploy, and test containerized applications with zero-downtime releases.

Course Objectives

- Introduction to deploying Cloud Native Applications on a Kubernetes cluster.

Audience

- Developers who wish to deploy cloud applications.
Administrators who are new to Kubernetes.
Architects who are considering using container technologies in software architectures.
Site Reliability Engineers who are considering using Kubernetes.

Prerequisites

- Experience running commands and navigating the file system on Linux, MacOS, and Windows.
Experience with web application development and corresponding technologies.

Orientation to the Classroom Environment

DO100a is a **Bring Your Developer Workstation (BYDW)** class, where you use your own internet-enabled system to access the shared OpenShift cluster. The following operating systems are supported:

- Red Hat Enterprise Linux 8 or Fedora Workstation 34 or later
- Ubuntu 20.04 LTS or later
- Microsoft Windows 10
- macOS 10.15 or later

BYDW System Requirements

<i>Attribute</i>	<i>Minimum Requirements</i>	<i>Recommended</i>
CPU	1.6 GHz or faster processor	Multi-core i7 or equivalent
Memory	8 GB	16 GB or more
Disk	10 GB free space HD	10 GB or more free space SSD
Display Resolution	1024x768	1920x1080 or greater

You must have permissions to install additional software on your system. Some hands-on learning activities in DO100a provide instructions to install the following programs:

- Git 2.18 or later (Git Bash for Windows systems)
- The Kubernetes CLI (`kubectl`) v1.21 or later
- Minikube (Optional)

You might already have these tools installed. If you do not, then wait until the day you start this course to ensure a consistent course experience.



Important

Hands-on activities also require that you have a personal account on GitHub and a public, free internet service.

BYDW Systems Support Considerations

Depending on your system, you might see differences between your command-line shell and the examples given in this course.

Red Hat Enterprise Linux or Fedora Workstation

- If you use Bash as the default shell, then your prompt might match the `[user@host ~]$` prompt used in the course examples, although different Bash configurations can produce different results.
- If you use another shell, such as `zsh`, then your prompt format will differ from the prompt used in the course examples.
- When performing the exercises, interpret the `[user@host ~]$` prompt used in the course as a representation of your system prompt.
- All the commands from the exercises should be functional.

Ubuntu

- You might find differences in the prompt format.
- In Ubuntu, your prompt might be similar to `user@host:~$`.
- When performing the exercises, interpret the `[user@host ~]$` prompt used in the course as a representation of your Ubuntu prompt.
- All the commands from the exercises should be functional.

macOS

- You might find differences in the prompt format.
- In macOS, your prompt might be similar to `host:~ user$`.
- When performing the exercises, interpret the `[user@host ~]$` prompt used in the course as a representation of your macOS prompt.
- All the commands from the exercises should be functional.
- You might need to grant execution permissions to the installed runtimes.

Microsoft Windows

- Windows does not support Bash natively. Instead, you must use PowerShell.
- In Windows PowerShell, your prompt should be similar to `PS C:\Users\user>`.
- When performing the exercises, interpret the `[user@host ~]$` Bash prompt as a representation of your Windows PowerShell prompt.
- For some commands, Bash syntax and PowerShell syntax are similar, such as `cd` or `ls`. You can also use the slash character (`/`) in file system paths.
- For other commands, the course provides help to transform Bash commands into equivalent PowerShell commands.
- This course only provides support for Windows PowerShell.
- The Windows firewall might ask for additional permissions in certain exercises.

Executing Long Commands

This course breaks long commands into multiple lines by using the backslash character (`\`), for example:


```
[user@host ~]$ long command \  
argument_1 \  
argument_2
```

The preceding example works on the Linux and macOS systems.

On Windows, use the backtick character (`), for example:

```
[user@host ~]$ long command `  
argument_1 `  
argument_2
```

Alternatively, you can type commands in one line on all systems, such as:

```
[user@host ~]$ long command argument_1 argument_2
```

Chapter 1

Introducing Containers and Kubernetes

Goal

Describing containers and container orchestration with Kubernetes.

Objectives

- Get an overview of what containers are, how they improve the software life cycle, and samples of different container runtimes.
- Recognize Kubernetes as a container orchestration tool.

Sections

- Introducing Containers and Container Engines
- Introducing Kubernetes and Container Orchestration

Introducing Containers and Container Engines

Objectives

After completing this section, you should be able to get an overview of what containers are, how they improve the software life cycle, and samples of different container runtimes.

Traditional Applications

Traditional software applications typically depend on other libraries, configuration files, or services that are provided by the runtime environment. The application runtime environment is a physical host or virtual machine (VM) and application dependencies are installed as part of the host.

For example, consider a Python application that requires access to a common shared library that implements the TLS protocol. A system administrator installs the required package that provides the shared library before installing the Python application.

The major drawback to a traditional deployment is that the application's dependencies are mixed with the runtime environment. Because of this, an application might break when any updates or patches are applied to the base operating system (OS).

For example, an update to the TLS shared library removes TLS 1.0 as a supported protocol. Updating the library breaks a Python application that is strictly dependent on TLS 1.0. The system administrator must downgrade the library to keep the application running, but this prevents other applications from using the benefits of the updated package.

To alleviate potential breakages, a company might maintain a full test suite to guarantee OS updates do not affect applications.

Furthermore, applications must be stopped before updating the associated dependencies. To minimize downtime, organizations use complex systems to provide high availability. Maintaining multiple applications on a single host often becomes cumbersome, and any update has the potential to break one of the applications.

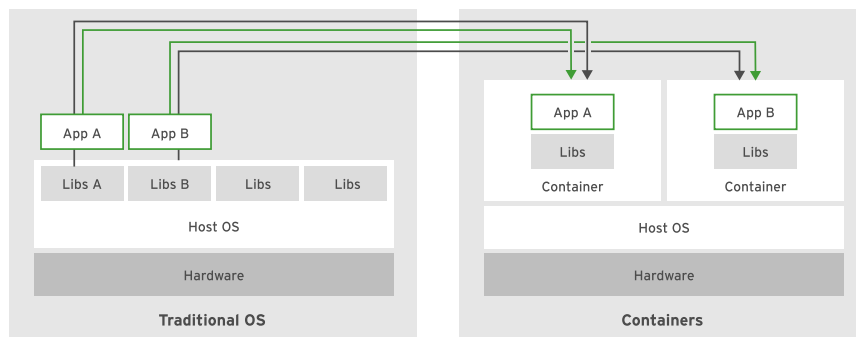


Figure 1.1: Container versus operating system differences

Containerized Applications

Deploying applications using containers is an alternative to the traditional methods. A container is a set of one or more processes that are isolated from the rest of the system.

Containers provide many of the same benefits as virtual machines, such as security, storage, and network isolation. Containers require fewer hardware resources and are quick to start and terminate. They also isolate the libraries and the runtime resources, such as CPU and storage, and minimize the impact of OS updates.

Beyond improving efficiency, elasticity, and reusability of hosted applications, container usage improves application portability. The Open Container Initiative (OCI) provides a set of industry standards that define a container runtime specification and a container image specification. The image specification defines the format for the bundle of files and metadata that form a container image. When you build a container image compliant with the OCI standard, you can use any OCI-compliant container engine to execute the contained application.

There are many container engines available to manage and execute containers, including Rocket, Drawbridge, LXC, Docker, and Podman.

The following are other major advantages to using containers:

Low hardware footprint

Containers use OS-internal features to create an isolated environment where resources are managed using OS facilities such as namespaces and cgroups. This approach minimizes the amount of CPU and memory overhead compared to a virtual machine hypervisor. Running an application in a VM isolates the application from the running environment, but it requires a heavy service layer to achieve the level of isolation provided by containers.

Environment isolation

Containers work in a closed environment where changes made to the host OS or other applications do not affect the container. Because the libraries needed by a container are self-contained, the application can run without disruption. For example, each application can exist in its own container with its own set of libraries. An update made to one container does not affect other containers.

Quick deployment

Containers deploy quickly because there is no need to install the entire underlying operating system. Normally, to support isolation, a host requires a new OS installation, and any update might require a full OS restart. A container restart does not require stopping any services on the host OS.

Multiple environment deployment

In a traditional deployment scenario using a single host, any environment differences could break the application. By using containers, all application dependencies and environment settings are encapsulated in the container image.

Reusability

The same container can be reused without the need to set up a full OS. For example, the same database container that provides a production database service can be used by each developer to create a development database during application development. By using containers, there is no longer a need to maintain separate production and development database servers. A single container image is used to create instances of the database service.

Often, a software application with all of its dependent services (databases, messaging, file systems) are made to run in a single container. This can lead to the same problems associated with traditional software deployments to virtual machines or physical hosts. In these instances, a multicontainer deployment might be more suitable.

Furthermore, containers are an ideal approach when using microservices for application development. Each service is encapsulated in a lightweight and reliable container environment

that can be deployed to a production or development environment. The collection of containerized services required by an application can be hosted on a single machine, removing the need to manage a machine for each service.

In contrast, many applications are not well suited for a containerized environment. For example, applications accessing low-level hardware information, such as memory, file systems, and devices may be unreliable due to container limitations.



References

Home - Open Containers Initiative

<https://www.opencontainers.org/>

Introducing Kubernetes and Container Orchestration

Objectives

After completing this section, you should be able to recognize Kubernetes as a container orchestration tool.

Limitations of Containers

Containers provide an easy way to package and run services. As the number of containers managed by an organization grows, the manual work of managing them grows disproportionately.

When using containers in a production environment, enterprises often require the following capabilities:

- Easy communication between a large number of services
- Resource limits on applications
- Ability to respond to application usage spikes by increasing or decreasing replicas
- Gradual rollout of a new release to different users

Enterprises often require a container orchestration technology because container runtimes, by themselves, do not adequately address the above requirements.

Kubernetes Overview

Kubernetes is a container orchestration platform that simplifies the deployment, management, and scaling of containerized applications.

A pod is the smallest manageable unit in Kubernetes, and consists of at least one container. Kubernetes also uses pods to manage the containers within and their resource limits as a single unit.

Kubernetes Features

Kubernetes offers the following features on top of a container engine:

Service discovery and load balancing

Kubernetes enables inter-service communication by assigning a single DNS entry to each set of containers. This way, the requesting service only needs to know the target's DNS name, allowing the cluster to change the container's location and IP address. This permits load-balancing requests across the pool of container replicas.

Horizontal scaling

Applications can scale up and down manually or automatically with a configuration set, by using either the command-line interface or the web UI.

Self-healing

Kubernetes can use user-defined health checks to monitor containers to restart and reschedule them in case of failure.

Automated rollout

Kubernetes can gradually release updates to your application's containers while checking their status. If something goes wrong during the rollout, Kubernetes can roll back to the previous version of the application.

Secrets and configuration maps

You can manage the configuration settings and secrets of your applications without rebuilding containers. Configuration maps store these settings in a way that decouples them from the pods and containers using them. Application secrets can include any configuration setting that must be kept private, such as user names, passwords, and service endpoints.

Operators

Operators are packaged Kubernetes applications that bring the knowledge of application lifecycles into the Kubernetes cluster. Applications packaged as Operators use the Kubernetes API to update the cluster's state by reacting to changes in the application state.



References

Production-Grade Container Orchestration - Kubernetes

<https://kubernetes.io/>

Summary

In this chapter, you learned:

- Applications running in containers are decoupled from the host operating system's libraries.
- Among other features, container orchestration platforms provide tooling to automate the deployment and management of application containers.

Chapter 2

Running Containerized Applications

Goal

Spin-up your first application in Kubernetes.

Objectives

- See the differences between several Kubernetes implementations, and understand how to prepare different Kubernetes flavours for this course.
- Review the basic usage of the kubectl command and understand how to connect to your Kubernetes cluster by using the CLI.
- Execute a pre-built application in your Kubernetes cluster and review the resources related to the process.

Sections

- Contrasting Kubernetes Distributions (and Guided Exercise)
- Introducing Kubectl (and Guided Exercise)
- Running and Interacting with Your First Application (and Guided Exercise)

Contrasting Kubernetes Distributions

Objectives

After completing this section, you should be able to see the differences between several Kubernetes implementations, and understand how to prepare different Kubernetes flavours for this course.

Kubernetes Distributions

Kubernetes has historically been a general solution for container management and orchestration. With this versatility, Kubernetes can solve the same problems in different ways depending on needs and opinions. Because of this, Kubernetes has evolved into different opinionated distributions based on:

- The target size of the cluster: From small single-node clusters to large-scale clusters of hundreds of thousands of nodes.
- The location of the nodes: Either locally on the developer workstation, on premises (such as a private data center), on the cloud, or a hybrid solution of those two.
- The ownership of the management: Self-managed clusters versus Kubernetes-as-a-Service.

The following table shows a classification for some of the most popular Kubernetes distributions:

	Big Scale	Small Scale
Self-Managed - Local		minikube, CodeReady Containers, Microk8s, Docker Kubernetes
Self-Managed - On Premises / Hybrid	Red Hat OpenShift, VMWare Tanzu, Rancher	
Kubernetes-as-a-Service - On Cloud	OpenShift Dedicated, Google Container Engine, Amazon EKS	Developer Sandbox



Note

This course supports minikube (version 1.20.0) for local development and Developer Sandbox for remote development. Instructions and exercises have been tested in the following operating systems:

- Fedora Linux 33 and 34
- Red Hat Enterprise Linux 8
- Windows 10 Pro and Enterprise
- MacOS Big Sur (11.3.1)

Visit the links in the *References* section for a comprehensive list of Kubernetes certified distributions.

Kubernetes Extensions

Kubernetes is highly extendable for adding more services to the platform. Each distribution provides different approaches (or none) for adding capabilities to Kubernetes:

DNS

DNS allows internal name resolution inside the cluster, so pods and services can refer to others by using a fixed name.

Both `minikube` and OpenShift include a `CoreDNS` controller that provides this feature.

Dashboard

The dashboard provides a graphical user interface to Kubernetes.

`minikube` provides an add-on and utility commands for using the general-purpose `Dashboard` open source application. OpenShift includes the `Console`, a dedicated application that integrates most of the Kubernetes extensions provided by OpenShift.

Ingress

The ingress extension allows traffic to get into the cluster network, redirecting requests from managed domains to services and pods. Ingress enables services and applications inside the cluster to expose ports and features to the public.

`minikube` uses an ingress add-on based on the `ingress-nginx` controller.



Note

You must install the ingress add-on for `minikube` for some exercises. Refer to *Guided Exercise: Contrasting Kubernetes Distributions* for instructions.

OpenShift deploys an ingress controller based on `HAProxy` and controlled by a `Ingress Operator`. OpenShift also introduces the `route` resource. A `route` extends the `ingress` manifest to ease controlling ingress traffic.

Storage

The storage extension allows pods to use persistent storage and nodes to distribute and share the storage contents.

OpenShift bases its storage strategy on `Red Hat OpenShift Data Foundation`, a storage provider supporting multiple storage strategies across nodes and hybrid clouds. `minikube` provides out-of-the-box storage by using the underlying storage infrastructure (either local the file system or the virtual machine's file-system). This feature is provided by the `storage-provisioner` add-on. `minikube` also provides a `storage-provisioner-gluster` add-on that allows Kubernetes to use `Gluster` as shared persistent storage.

Authentication and authorization.

Kubernetes embeds a certificate authority (CA) and considers anyone that presents a certificate issued by that CA as a valid user.

`minikube` provides the user with an administrator `minikube` account, so users have total control over the cluster.

Different OpenShift implementations differ on authentication features, but all of them agree on avoiding the use of administration accounts. Developer Sandbox provides limited access to the user, restricting them to the `username-dev` and `username-stage` namespaces.

Authorization in Kubernetes is role based. Authorized users or administrators can assign predefined roles to users on each resource. For example, administrators can grant read-only access to auditor users to application namespaces.

Operators

Operators are a core feature of most Kubernetes distributions. Operators allow automated management of applications and Kubernetes services, by using a declarative approach.

`minikube` requires the `olm` add-on to be installed to enable operators in the cluster.

OpenShift distributions enable operators by default, despite the fact that Kubernetes-as-a-Service platforms usually restrict user-deployed operators. Developer Sandbox does not allow users to install operators, but comes with the `RHOAS-Operator` and the `Service Binding Operator` by default.

Comparison summary of Kubernetes features

	minikube	Developer Sandbox
DNS		
Dashboard	Dashboard add-on	OpenShift Console
Ingress	NGINX Ingress add-on	Operator-controlled HAProxy
Storage	Local or Gluster add-ons	Red Hat OpenShift Data Foundation,
Authentication	Administrator <code>minikube</code> user	Developer used restricted to 2 namespaces
Operators	OLM add-on. No restrictions	Limited to <code>RHOAS</code> and <code>Service Binding</code>



References

minikube documentation

<https://minikube.sigs.k8s.io/docs/>

Developer Sandbox

<https://developers.redhat.com/developer-sandbox>

CNCF Cloud Native Interactive Landscape - Certified Kubernetes Platforms

<https://landscape.cncf.io/card-mode?category=certified-kubernetes-distribution,certified-kubernetes-hosted,certified-kubernetes-installer&grouping=category>

Certificate Signing Requests

<https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#normal-user>

► Guided Exercise

Contrasting Kubernetes Distributions

In this exercise you will prepare your development environment to use a local or remote Kubernetes instance.

Outcomes

You should be able to:

- Install a local Kubernetes instance by using `minikube` on Linux, macOS or Windows.
- Register for using a remote Kubernetes instance by using `Developer Sandbox for Red Hat OpenShift`.

Instructions



Note

Installing a local Kubernetes cluster requires administrative privileges in your development workstation. If you do not have administrative privileges then jump directly to *Guided Exercise: Contrasting Kubernetes Distributions* to use a remote Kubernetes cluster.

Deploying a fully developed, multi-node Kubernetes cluster typically requires significant time and compute resources. With `minikube`, you can quickly deploy a local Kubernetes cluster, allowing you to focus on learning Kubernetes operations and application development.

`minikube` is an open source utility that allows you to quickly deploy a local Kubernetes cluster on your personal computer. By using virtualization technologies, `minikube` creates a *virtual machine* (VM) that contains a single-node Kubernetes cluster. VMs are virtual computers and each VM is allocated its own system resources and operating system.

The latest `minikube` releases also allow you to create your cluster by using containers instead of virtual machines. Nevertheless, this solution is still not mature, and it is not supported for this course.

`minikube` is compatible with Linux, macOS, and Windows.

To install `minikube` on your system, you will need:

- An Internet connection
- At least 2 GB of free memory
- 2 CPUs or more
- At least 20 GB of free disk space
- A locally installed hypervisor (using a container runtime is not supported in this course)

Before installing `minikube`, a hypervisor technology must be installed or enabled on your local system. A **hypervisor** is software that creates and manages virtual machines (VMs) on a shared physical hardware system. The hypervisor pools and isolates hardware resources for VMs, allowing many VMs to run on a shared physical hardware system, such as a server.

► 1. Using minikube in Linux-based systems

In this course we support Fedora Linux 33 and 34, as well as Red Hat Enterprise Linux 8.

1.1. Installing a Hypervisor on Linux

The preferred hypervisor for Linux systems is `kvm2` [<https://minikube.sigs.k8s.io/docs/drivers/kvm2/>]. `minikube` communicates with the hypervisor using the `libvirt` virtualization API libraries.



Note

Prefix the following commands with `sudo` if you are running a user without administrative privileges.

Use your system package manager to install the complete set of virtualization libraries:

```
[root@host ~]# dnf install @virtualization
```

or select the minimum set of required libraries

```
[root@host ~]# dnf install qemu-kvm libvirt libvirt-python libguestfs-tools
virt-install
```

Start the `libvirtd` service:

```
[root@host ~]# systemctl start libvirtd
...output omitted...
[root@host ~]# systemctl enable libvirtd
...output omitted...
```

1.2. Installing minikube on Linux

There are three alternatives to install `minikube` in a Linux system:

- If your system contains a package manager or software manager including `minikube`, then use it and verify the version matches the minimum requirements.

```
[root@host ~]# dnf install minikube
```

- If the repositories for your package manager do not include an appropriate version for `minikube`, then go to <https://github.com/kubernetes/minikube/releases> and download the latest release matching your operating system.

For Debian-based systems, download the file `minikube_VERSION_[amd64|arm64|armhf|ppc64le|s390x].deb` and install it using the `dpkg` command:

```
[root@host ~]# dpkg -i FILE
```

For RPM-based distributions, download the file `minikube-VERSION.[aarch64|armv7hl|ppc64le|s390x|x86_64].rpm` and install it using the `rpm` command:

```
[root@host ~]# rpm -Uvh FILE
```

- Alternatively, download the binary `minikube-linux-[amd64|arm|arm64]` file and install using the `install` command:

```
[root@host ~]# install FILE /usr/local/bin/minikube
```

1.3. Starting Your minikube Cluster on Linux

To initialize your `minikube` cluster, use the `minikube start` command.

```
[root@host ~]# minikube start --driver=kvm2
# minikube v1.20.0 on Fedora 33
# Using the kvm2 driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating kvm2 VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  ▪ Generating certificates and keys ...
  ▪ Booting up control plane ...
  ▪ Configuring RBAC rules ...
# Verifying Kubernetes components...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```



Note

To set the default driver, run the command `minikube config set driver DRIVER`.

▶ 2. Using minikube on macOS systems

This course supports macOS Big Sur (version 11.3.1) and later minor updates.

2.1. Installing a Hypervisor on macOS

Despite Docker being the preferred installation method in macOS, current `minikube` versions do not support all the features required for this course when running the `docker` driver (to be precise, `ingress` add-on is not yet supported). For this reason, we are using the `Oracle VM VirtualBox` hypervisor for running the virtual machines needed to support `minikube` on macOS.

Oracle VM VirtualBox is a free and open source hypervisor available for macOS systems. To install Oracle VM VirtualBox:

1. Download the latest version of VirtualBox for OS X hosts from <https://virtualbox.org/wiki/Downloads>
2. Open the downloaded `dmg` file and follow the onscreen instructions to complete the installation.



Note

Network connectivity might be temporarily lost while VirtualBox installs virtual network adapters. A system reboot can also be required after a successful installation.

Alternatively, if the `brew` command is available in your system, then you can install VirtualBox using the `brew install` command.

```
host:~ root# brew install virtualbox
```

2.2. Installing minikube on macOS

To install `minikube` on macOS, download from <https://github.com/kubernetes/minikube/releases> the appropriate file for your architecture: `minikube-darwin-amd64` or `minikube-darwin-arm64`.

Then open a terminal window, change to the directory where you downloaded the installer and use the `install` command with administrator privileges to run the installer. Make sure you install `minikube` in a folder in your system path, such as `/usr/local/bin/minikube`

```
host:~ root# cd Downloads
host:~ root# sudo install _FILE /usr/local/bin/minikube
```

This places the `minikube` executable in `/usr/local/bin/minikube`.

Apple notarizing features forbid running files downloaded from the internet unless authorized. If you try to run the `minikube` command then you will get a "minikube" cannot be opened because the developer cannot be verified message, and the application will be terminated.

To authorize the `minikube` application, use the `xattr` command with administrative privileges with the following options:

```
host:~ root# sudo xattr -r -d com.apple.quarantine /usr/local/bin/minikube
```

Verify that now you can execute the `minikube` command:

```
host:~ root# minikube version
minikube version: v1.20.0
commit: c61663e942ec43b20e8e70839dcca52e44cd85ae
```

Your output can differ, but must show the available version and the commit it is based on.

2.3. Starting Your minikube Cluster on macOS

To initialize your `minikube` cluster, use the `minikube start` command.

```
host:~ root# minikube start --driver=virtualbox
# minikube v1.20.0 on Darwin 11.3.1
# Using the virtualbox driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating virtualbox VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  ▪ Generating certificates and keys ...
  ▪ Booting up control plane ...
  ▪ Configuring RBAC rules ...
# Verifying Kubernetes components...
```

```

▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
  namespace by default

```

**Note**

To set the default driver, run the command `minikube config set driver DRIVER`.

▶ 3. Using minikube on Windows

In this course we support Windows 10 Pro and Enterprise versions.

3.1. Installing a Hypervisor on Windows

There are several hypervisors available for Windows systems, including Oracle VM VirtualBox and Microsoft Hyper-V.

**Warning**

System driver conflicts might occur if more than one hypervisor is installed or enabled. Do not install or enable more than one hypervisor on your system.

Oracle VM VirtualBox installation

Oracle VM VirtualBox is a free, open source hypervisor. As the original driver for minikube, Oracle VM VirtualBox provides the most stability for Microsoft Windows 10 users.

1. Download the latest version of VirtualBox for Windows Hosts from <https://virtualbox.org/wiki/Downloads>
2. Open the downloaded VirtualBox executable and follow the onscreen instructions to complete the installation.

**Note**

Network connectivity might be temporarily lost while VirtualBox installs virtual network adapters. A system reboot can also be required after a successful installation.

Enabling Microsoft Hyper-V

Microsoft Hyper-V is a built-in hypervisor available on modern 64-bit versions of Microsoft Windows 10 Pro, Enterprise, and Education editions. Enabling Microsoft Hyper-V can be accomplished through PowerShell or by adjusting system Settings.

Refer to Microsoft Hyper-V documentation for system and hardware requirements.

- Via PowerShell
 - Launch a PowerShell console as Administrator.
 - In the console window, run the following command:

```
PS C:\> Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

- Restart your system when prompted to finish the installation process.
- Via Settings
 - In the search box on the taskbar, type **Programs and Features**, and select it from the search results.
 - Select **Turn Windows features on or off** from the list of options under Control Panel Home.
 - Select **Hyper-V** and click **OK** to begin the installation.
 - Restart your system when prompted to finish the installation process.

With Microsoft Hyper-V successfully enabled, create an external virtual switch to grant **minikube** access to your physical network.

1. Open a PowerShell console as Administrator.
2. Determine the name of the network adapter, such as Wi-Fi or Ethernet, to use by running `Get-NetAdapter`.
3. Create an external virtual switch named **minikube** that uses the selected network adapter and allows the management operating system to share the adapter:

```
PS C:\> New-VMSwitch -name minikube -NetAdapterName <AdapterName>
-AllowManagementOS $true
```

4. Restart your system to clear out potential routing problems.

3.2. Installing minikube on Windows

With a hypervisor installed, your system is now ready to begin the **minikube** installation process.

1. Download the stand-alone **minikube** Windows installer from <https://github.com/kubernetes/minikube/releases>
2. Open the downloaded **minikube-installer.exe** to begin the guided installation process.



Note

If you executed the **minikube-installer.exe** installer from a terminal window, close the terminal and open a new one before you start using **minikube**.

3.3. Starting Your minikube Cluster on Windows

To initialize your **minikube** cluster, use the **minikube start** command.

To use the Microsoft Hyper-V driver, provide the `--driver=hyperv` option to **minikube**:

```

PS C:\> minikube start --driver=hyperv
# minikube v1.20.0 on Microsoft Windows 10 Enterprise 10.0.18363 Build 18363
# Using the hyperv driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating hyperv VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  ▪ Generating certificates and keys ...
  ▪ Booting up control plane ...
  ▪ Configuring RBAC rules ...
# Verifying Kubernetes components...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default

```

To use the Oracle VM VirtualBox driver, provide the `--driver=virtualbox` option to `minikube`:

```

PS C:\> minikube start --driver=virtualbox
# minikube v1.20.0 on Microsoft Windows 10 Enterprise 10.0.18363 Build 18363
# Using the virtualbox driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating virtualbox VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  ▪ Generating certificates and keys ...
  ▪ Booting up control plane ...
  ▪ Configuring RBAC rules ...
# Verifying Kubernetes components...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default

```



Note

To set the default driver, run the command `minikube config set driver DRIVER`.

► 4. Verifying your minikube installation

Use the `minikube status` command to validate that the `minikube` installation is running successfully:

```
[root@host ~]# minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

In case of errors, make sure you are using the appropriate driver during the installation, or refer to minikube Get Started documentation [<https://minikube.sigs.k8s.io/docs/start/>] for troubleshooting.

► 5. Adding extensions

`minikube` comes with the bare minimum set of features. To add more features, `minikube` provides an add-on based extension system. Developers can add more features by installing the needed add-ons.

Use the `minikube addons list` command for a comprehensive list of the add-ons available and the installation status.

- Installing the Ingress Add-on. For this course you must install the `ingress` add-on.

With your cluster up and ready, use the following command to enable the add-on:

```
[root@host ~]# minikube addons enable ingress
  ▪ Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0
  ▪ Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
  ▪ Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
# Verifying ingress addon...
# The 'ingress' addon is enabled
```

Versions and docker images can vary in your deployment, but make sure the final validation is successful.

- Installing the Dashboard add-on. The `dashboard` add-on is not required for this course but serves as a visual graphical interface if you are not comfortable with CLI commands.

Enable the `dashboard` add-on with the following command:

```
[root@host ~]# minikube addons enable dashboard
  ▪ Using image kubernetesui/dashboard:v2.1.0
...output omitted...
# The 'dashboard' addon is enabled
```

Once the dashboard is enabled you can reach it by using the `minikube dashboard` command. This command will open the dashboard web application in your default browser.

Press `Ctrl+C` in the terminal to finish the connection to the dashboard.

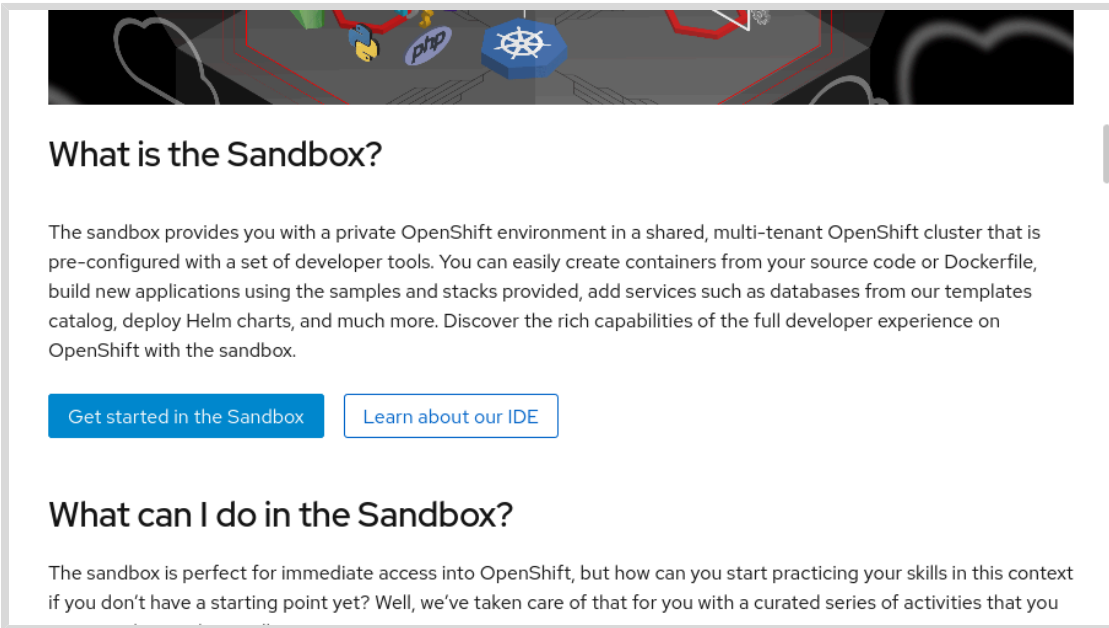
▶ 6. Using a Developer Sandbox for Red Hat OpenShift as a Remote Kubernetes cluster

Developer Sandbox for Red Hat OpenShift is a free Kubernetes-as-a-Service platform offered by Red Hat Developers, based on Red Hat OpenShift.

Developer Sandbox allows users access to a pre-created Kubernetes cluster. Access is restricted to two namespaces (or projects if using OpenShift nomenclature). Developer Sandbox deletes pods after eight consecutive hours of running, and limits resources to 7 GB of RAM and 15 GB of persistent storage.

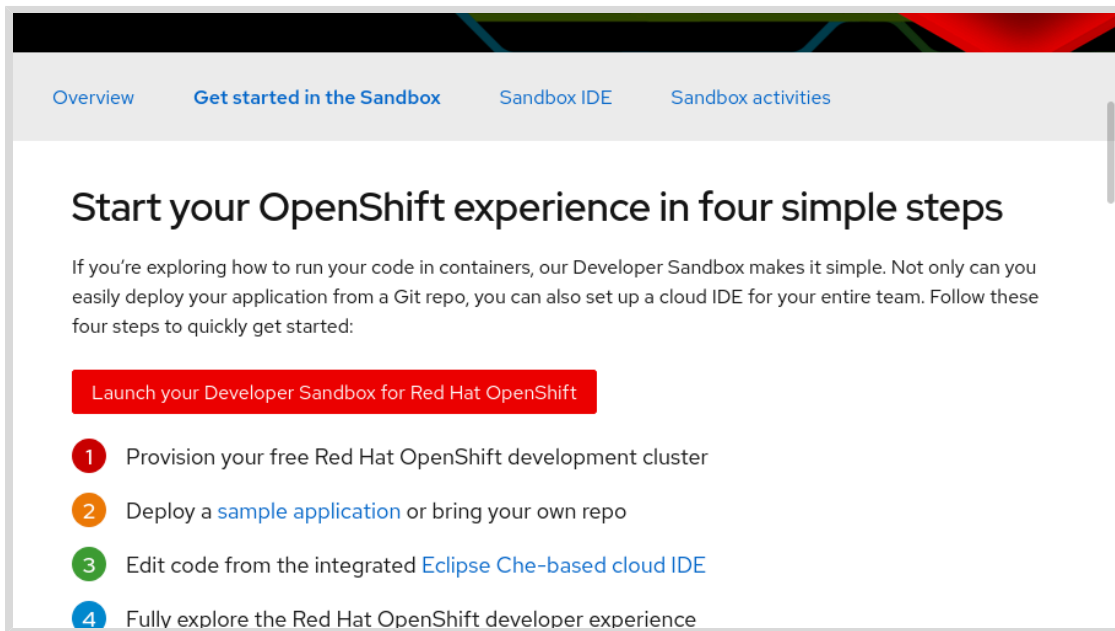
6.1. Create a Developer Sandbox account

Go to <https://developers.redhat.com/developer-sandbox> and click **Get started in the Sandbox**.

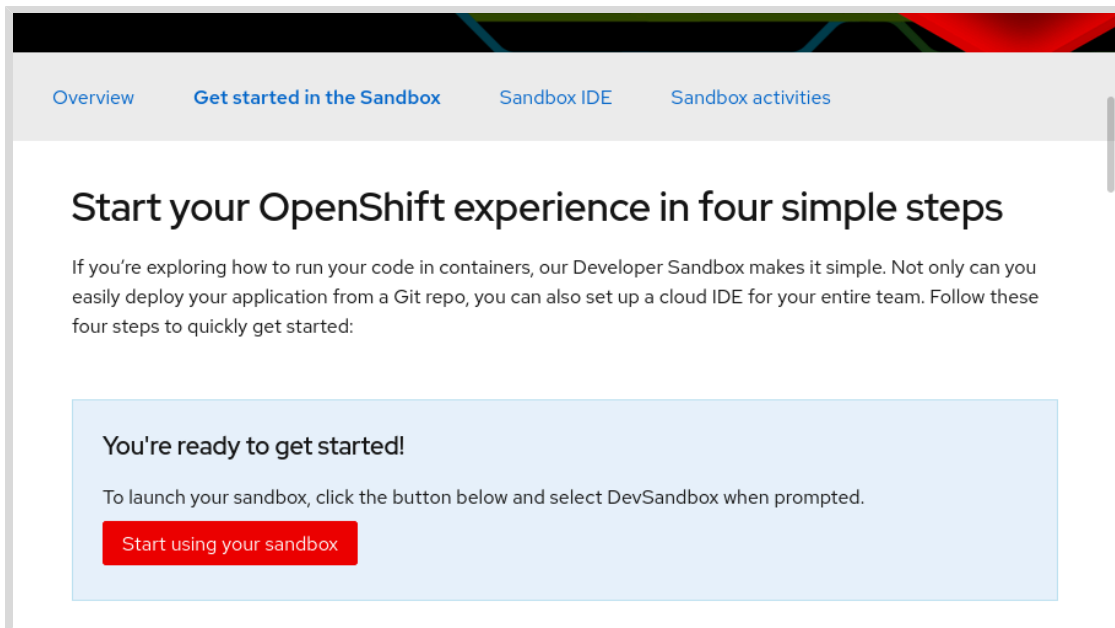


You need a free Red Hat account to use Developer Sandbox. Log in to your Red Hat account, or if you do not have one, then click **Create one now**. Fill in the form choosing a **Personal** account type, and then click **CREATE MY ACCOUNT**. You might need to accept Red Hat terms and conditions to use the Developer Program services.

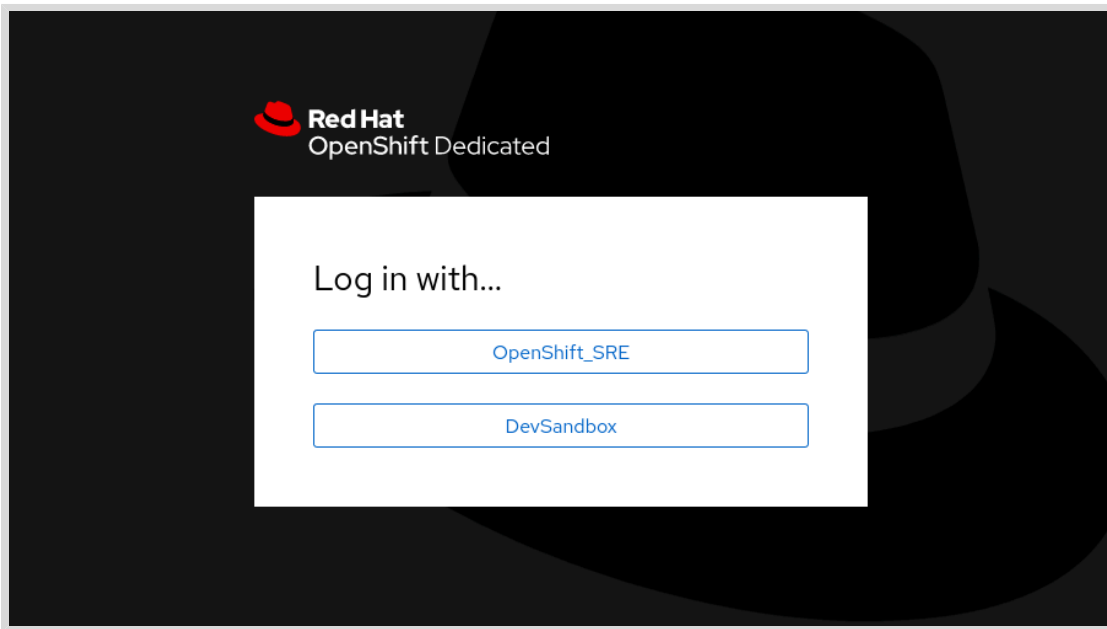
When the account is ready you will be redirected back to the Developer Sandbox page. Click **Launch your Developer Sandbox for Red Hat OpenShift** to log in to Developer Sandbox.



If you just created your account, then you might need to wait some seconds for account approval. You might need to verify your account via 2-factor authentication. Once the account is approved and verified, Click **Start using your sandbox**. You might need to accept Red Hat terms and conditions to use the Developer Sandbox.



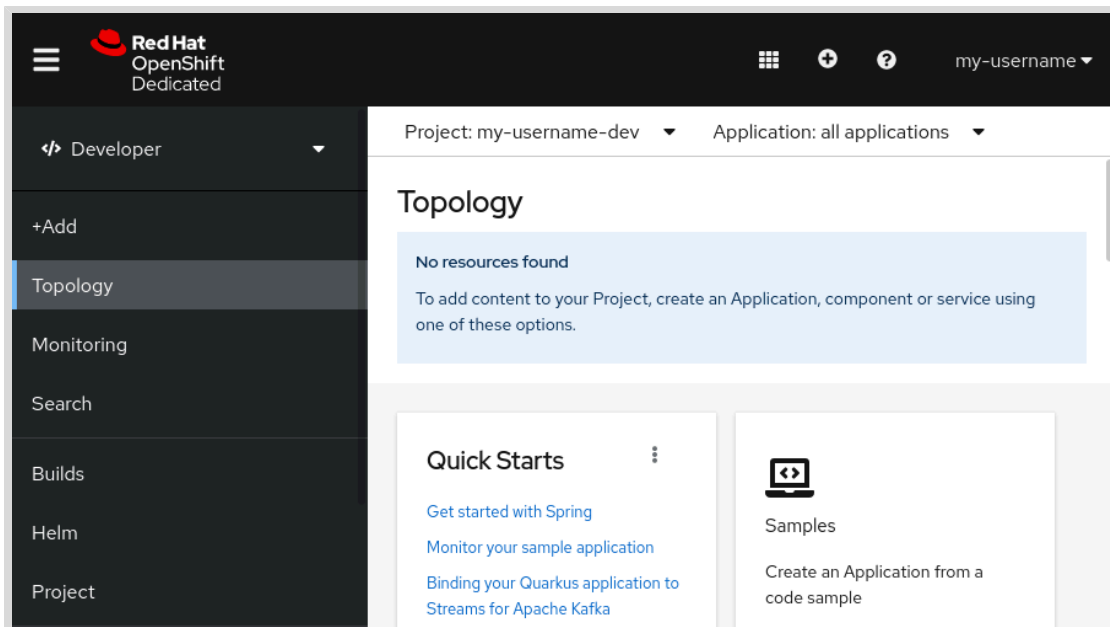
In the OpenShift log in form, click **DevSandbox** to select the authentication method.



If requested, then log in using your Red Hat account credentials.

6.2. Start Using Developer Sandbox

The Kubernetes cluster and two namespaces are created for you. Only the *username-dev* and *username-stage* namespaces are available, and you can not create more.



7. Enabling external access to Ingress.

Some exercises require you to identify the external IP and hostname associated to your Ingress so you can access your application from outside the cluster.

7.1. External access to Minikube.

Routing traffic from your local machine to your Minikube Kubernetes cluster requires two steps.

First you must find the local IP assigned to your Ingress add on. The `minikube ip` command is the easiest way to find the ingress IP:

```
[user@host ~]$ minikube ip
192.168.99.103
```

Your IP will probably be different as it depends on your virtual environment configuration.

Second, you must declare a hostname for your ingress, and associate the hostname to the ingress IP. For this course, unless the hostname is already in use, you will declare `hello.example.com` as the hostname.

The association between an IP and a hostname is usually performed by a DNS server, but to avoid registering a domain name to use with Minikube, we can use the system's local name resolution service.

In **Linux and macOS** systems, edit the `/etc/hosts` file with elevated privileges. You might use the `sudo` command or the `su` command to obtain elevated privileges. In **Windows** systems, edit the `C:\Windows\System32\drivers\etc\hosts` file with administrative privileges.

Add the following line to the bottom of the file and replace `<IP-ADDRESS>` with the IP address listed in the previous step.

```
<IP-ADDRESS> hello.example.com
```



Note

If for any reason you need to delete and recreate your Minikube cluster, then review the IP address assigned to the cluster and update the hosts file accordingly.

For accessing services in the cluster you will use the declared hostname and potentially any path associated to the ingress. So, if using the `hello.example.com` hostname and assuming the application is mapped to the path `/myapp`, then your application will be available in the URL `http://hello.example.com/myapp`.

7.2. External access to Developer Sandbox.

External access to Developer Sandbox is already prepared. The ingress controller in Developer Sandbox is pre-configured to listen to requests sent to any sub-domain for a hostname known as the `wildcard domain`. The wildcard domain is a configuration value set by administrators when creating the cluster.

To enable external access to a service you must provide a sub-domain for the wildcard domain, so first you must know the wildcard domain.

There are several approaches for finding the wildcard domain associated to your Developer Sandbox instance:

Infer the wildcard domain from the Console URL

When you log into your Developer Sandbox account, you will be redirected to the OpenShift Console installed in the cluster. The URL for the OpenShift Console is of the form `https://console-openshift-console.apps._WILDCARD-DOMAIN_`.

To get the wildcard domain, remove from the API URL the `https://console-openshift-console`, and replace `api` by `apps`. For example, the wildcard domain for the Console URL `https://console-openshift-console.apps.sandbox.x8i5.p1.openshiftapps.com` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

Infer the wildcard domain from the API URL

In *Guided Exercise: Connecting Kubectl To Your Cluster* you will find instructions to identify the API URL for your Developer Sandbox cluster.

To get the wildcard domain, remove from the API URL the `https://`, the `:6443`, and change `api` to `apps`. For example, the wildcard domain for the API URL `https://api.sandbox.x8i5.p1.openshiftapps.com:6443` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

Infer from a dry-run route creation

If you are using a Linux or macOS system, you can use the `kubectl` command to create a route resource that contains an example URL and extract the hostname from it:

```
[user@host ~]$ echo '{"apiVersion": "route.openshift.io/v1", \
"kind": "Route","metadata": {"name": "example"}, \
"spec": {"to": {"kind": "Service","name": "hello"}}}' | kubectl apply \
--dry-run=server -o jsonpath='{.spec.host}' -f -
example-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com
```

To get the wildcard domain, remove the first part of the hostname, that is everything before the first period. For example, the wildcard domain for the hostname `example-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

Once you know the wildcard domain for your Developer Sandbox cluster, use it to generate a sub-domain to be used by your services. Remember that sub-domains must be unique for the shared Developer Sandbox cluster. One method for creating a unique sub-domain is to compose it in the format of `<DEPLOYMENT-NAME>-<NAMESPACE-NAME>.<WILDCARD-DOMAIN>`.

So, if using the `apps.sandbox.x8i5.p1.openshiftapps.com` wildcard domain and assuming a deployment named `hello` in a namespace named `username-dev` then you can compose your application hostname as `hello-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com`.

Assuming the application is mapped to the path `/myapp`, then your application will be available in the URL `http://hello-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com/myapp`.

Finish

This concludes the guided exercise.

Introducing Kubectl

Objectives

After completing this section, you should be able to review the basic usage of the kubectl command and understand how to connect to your Kubernetes cluster by using the CLI.

Introducing kubectl

The `kubectl` tool is a Kubernetes command-line tool that allows you to interact with your Kubernetes cluster. It provides an easy way to perform tasks such as creating resources or redirecting cluster traffic. The `kubectl` tool is available for the three main operating systems (Linux, Windows and macOS).

For example, the following command displays the kubectl and Kubernetes version.

```
[user@host ~]$ kubectl version
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
  GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
  BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
  Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20",
  GpodsitVersion:"v1.20.0+75370d3",
  GitCommit:"75370d3fb99594d6f0263f3de0bd08237381b77d", GitTreeState:"clean",
  BuildDate:"2021-05-09T17:55:51Z", GoVersion:"go1.15.7", Compiler:"gc",
  Platform:"linux/amd64"}
```

Introducing kubectl configuration

Kubectl reads all the information necessary to connect to the Kubernetes cluster from a config file within your system. By default, this file is located at `$HOME/kube/config`. You can change this path by setting the environment variable `KUBECONFIG` to a custom file.

For example, the following sample sets the `KUBECONFIG` to the file `/tmp/config`.

```
[user@host ~]$ export KUBECONFIG=/tmp/config
```

All the commands related to the `kubectl` configuration are of the form:

```
[user@host ~]$ kubectl config option
```

If you want to see what the configuration file contains, then you can use the following command.

```
[user@host ~]$ kubectl config view
```

The kubectl configuration file comprehends three topics:

- Cluster: the URL for the API of a Kubernetes cluster. This URL identifies the cluster itself.

- User: credentials that identify a user connecting to the Kubernetes cluster.
- Context: puts together a cluster (the API URL) and a user (who is connecting to that cluster).

For example, you might have two contexts that are using different clusters but the same user.

Defining Clusters

It is often necessary to work with multiple clusters, so `kubectl` can hold the information of several Kubernetes clusters. In relation to the configuration for `kubectl`, a cluster is just the URL of the API of the Kubernetes cluster. The `kubectl config set-cluster` command allows you to create a new cluster connection by using the API URL.

For example, the following command creates a new cluster connection named `my-cluster` with server `127.0.0.1:8087`.

```
[user@host ~]$ kubectl config set-cluster my-cluster --server=127.0.0.1:8087
```

Use the `get-clusters` option to list all available clusters.

```
[user@host ~]$ kubectl config get-clusters
my-cluster
my-cluster-2
```

Defining Users

The cluster configuration tells `kubectl` where the Kubernetes cluster is. The user configuration identifies who connects to the cluster. To connect to the cluster, it is necessary to provide an authentication method. There are several options to authenticate with the cluster:

- Using a token

The following command creates a new user named `my-user` with the token `Py93bt12mT`.

```
[user@host ~]$ kubectl config set-credentials my-user --token=Py93bt12mT
```

- Using basic authentication

The following command creates a new user named `my-user` with username `kubernetes-username` and password `kubernetes-password`.

```
[user@host ~]$ kubectl config set-credentials my-user \
--username=redhat-username --password=redhat-password
```

- Using certificates

The following command creates a new user named `my-user` with a certificate `redhat-certificate.crt` and a key `redhat-key.key`.

```
[user@host ~]$ kubectl config set-credentials my-user \
--client-certificate=redhat-certificate.crt \
--client-key=redhat-key.key
```

Use the `get-users` option to list all available users.

```
[user@host ~]$ kubectl config get-users
my-user
my-user-2
```

Defining Contexts

A context puts together a cluster and a user. Kubectl uses both to connect and authenticate against a Kubernetes cluster.

For example, the following command creates a new context by using a cluster named `my-cluster` and a user named `my-user`.

```
[user@host ~]$ kubectl config set-context \
--cluster=my-cluster --user=my-user
```

In a kubectl context, it is possible to set a namespace. If provided, then any command would be executed in that namespace. The following command creates a context that points to the `redhat-dev` namespace.

```
[user@host ~]$ kubectl config set-context my-context \
--cluster=my-cluster --user=my-user --namespace=redhat-dev
```

Once a context has been created, you can select it by using the `use-context` command.

```
[user@host ~]$ kubectl config use-context my-context
```

After executing the previous command, further kubectl commands will use the `my-cluster` context and, therefore, the cluster and user associated to that context.

You can also list the contexts available in the configuration by using the `get-contexts` option. The `*` in the `CURRENT` column indicates the context that you are currently using.

```
[user@host ~]$ kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	my-context	172.0.7.2:6443	my-user	redhat-dev
	my-context-2	172.1.8.0:6443	my-user-2	

Another way of checking the current context is by using the `current-context` option.

```
[user@host ~]$ kubectl config current-context
my-context
```

Working with resources

Once you are connected to a Kubernetes cluster, kubectl allows you to list, create, update and delete Kubernetes resources. Most of these commands will be introduced in later chapters in the course, but there are some of them that can be mentioned at this point.

The get command

This commands is used to display one or more resources.

For example, `kubectl get pods` will display all pods in the current namespace.

```
[user@host ~]$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
example1	1/1	Running	0	67s
example2	1/1	Running	0	67s

If you want to display just the information for one pod, then add the pod's name to the previous command.

```
[user@host ~]$ kubectl get pods example1
```

NAME	READY	STATUS	RESTARTS	AGE
example1	1/1	Running	0	67s

You can use this command to display other resources (services, jobs, ingresses...).



Note

Use the command `kubectl api-resources` to display all resource types that you can create.

The delete command

This command allows you to delete a resource.

For example, `kubectl delete pod example1` deletes the pod named `example1`.

```
[user@host ~]$ kubectl delete pod example1
```

pod "example1" deleted

You can use this command to delete other resources (services, jobs, ingresses...).

The apply command

A common way to manage Kubernetes resources is by using a **manifest**. A **manifest** is a YML or JSON file containing one or many Kubernetes resources.

The `apply` command allows you to create, update or delete resources from a manifest.

For example, the following snippet creates a **Deployment** resource.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
```



```
metadata:
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

If the snippet was in a file named `deployment.yml`, then you could use `apply` to create the deployment. Note that the `-f` option is used to indicate the file.

```
[user@host ~]$ kubectl apply -f deployment.yml
deployment.apps/nginx-deployment created
```



References

Overview of Kubectl

<https://kubernetes.io/docs/reference/kubectl/overview/>

Kubectl Command Reference

<https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands>

► Guided Exercise

Connecting Kubectl To Your Cluster

In this exercise you will install the kubectl command-line tool on your computer, and connect to the Kubernetes cluster that you will be using throughout the course.

Outcomes

You should be able to:

- Install kubectl
- Connect to Minikube (if you are using Minikube)
- Connect to the OpenShift Developer Sandbox (if you are using the OpenShift Developer Sandbox)

Before You Begin

Ensure you have either installed Minikube or created an OpenShift Developer Sandbox account.

Instructions

The installation procedure of kubectl depends on your operating system.

► 1. Installing kubectl in Linux-based systems.

The Linux distributions supported in this course are Fedora Linux 34 and Red Hat Enterprise Linux 8.

You can install kubectl by downloading the binary and moving it to your PATH. At the same time, it is possible to use a package-manager.

1.1. Using curl and the binary file

- Open a command-line terminal to download the kubectl binary.

```
[user@host ~]$ curl -LO https://dl.k8s.io/release/v1.21.0/bin/linux/amd64/kubectl
```

- Copy the binary to your PATH and make sure it has executable permissions.

```
[user@host ~]$ sudo install -m 0755 kubectl /usr/local/bin/kubectl
```

- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

1.2. Using the dnf package manager.

- Open a command-line terminal. Install the package by using dnf.

```
[user@host ~]$ dnf install kubectl
...output omitted...

=====
Package           Architecture      Version           Repository        Size
=====
Installing:
kubectl            x86_64            1.21.1-0          kubernetes        9.8 M

Transaction Summary
=====
Install 1 Package
...output omitted...
```

- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

▶ 2. Installing kubectl in macOS.

2.1. Using curl and the binary file

- Open a command-line terminal. Download the kubectl binary by using curl. Replace the `os-architecture` parameter depending on your Mac processor. If your Mac comes with an Intel processor, use `amd64`. If your Mac comes with an Apple Silicon processor, use `arm64`.

```
[user@host ~]$ curl -LO "https://dl.k8s.io/release/v1.21.0/bin/darwin/os-
architecture/kubectl"
```

- Give the binary file executable permissions. Move the binary file executable to your PATH.

```
[user@host ~]$ chmod +x ./kubectl
[user@host ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
```

- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

2.2. Using the homebrew package manager



Note

If you have previously installed Minikube with `homebrew`, `kubectl` should already be installed in your computer. You can skip the installation step and directly verify that `kubectl` has been installed correctly.

- Install `kubectl` by using `brew`

```
[user@host ~]$ brew install kubectl
...output omitted...
```

- Verify that `kubectl` has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
  GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
  BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
  Platform:"linux/amd64"}
```

▶ 3. Installing kubectl in Windows

3.1. Using the binary

- Create a new folder, such as `C:\kube`, to use as the destination directory of the `kubectl` binary download.
- Download the latest release of the `kubectl` binary from <https://dl.k8s.io/release/v1.21.0/bin/windows/amd64/kubectl.exe> and save it to the previously created folder.
- Add the binary to your `PATH`.
 - In the search box on the taskbar, type `env`, and select `Edit the system environment variables` from the search results.
 - Click `Environment Variables` on the `System Properties` screen.
 - Under the `System variables` section, select the row containing `Path` and click `Edit`. This will open the `Edit environment variable` screen.
 - Click `New` and type the full path of the folder containing the `kubectl.exe` (for example, `C:\kube`).
 - Click `OK` to save the change and close the editor.
 - Click `OK` → `OK` to close out of the remaining screens.
- Verify that `kubectl` has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

▶ 4. Download the D0100x-apps repository

The D0100x-apps GitHub repository contains files used throughout the course. The best way to work with it is by using Git. However, if you are not used to Git, you can simply download it as a regular folder and place it anywhere in your computer.

4.1. Using Git

If you have Git installed in your system, you can simply clone the repository.

```
[user@host ~]$ git clone https://github.com/RedHatTraining/D0100x-apps.git
...output omitted...
```

4.2. Downloading it from the GitHub web page

- Open a web browser and navigate to <https://github.com/RedHatTraining/D0100x-apps>.
- Click **Code** and then click **Download ZIP**. A ZIP file with the repository content is downloaded.

▶ 5. Connecting kubectl to the Kubernetes cluster

The D0100x-apps contains a script that handles all configurations for you under the `setup` directory. The script you should run depends on your operating system (Linux, macOS or Windows) and the Kubernetes distribution you use (Minikube or the OpenShift Developer Sandbox).

If you run the Minikube script, it will configure Minikube to work as a Kubernetes cluster with restricted access. You will only have access to two namespaces. This way, we simulate a real Kubernetes cluster, where usually developers do not have full access.



Note

If you want to recover full access over your cluster, then you can change the kubectl context to the default Minikube context, `minikube`. Use the command `kubectl config use-context minikube`.

If you run the OpenShift Developer Sandbox script, it will configure kubectl to run commands against the Openshift Developer Sandbox cluster. The script will ask you to provide some information such as cluster url, username or token.

5.1. Using Minikube

• Linux and macOS

In your command-line terminal, move to the D0100x-apps directory and run the script located at `./setup/operating-system/setup.sh`. Replace `operating-`

system for linux if you are using Linux. Use macOS if you are using macOS. Make sure the script has executable permissions.

```
[user@host D0100x-apps]$ chmod +x ./setup/operating-system/setup.sh
[user@host D0100x-apps]$ ./setup/operating-system/setup.sh
Creating namespace 'user-dev' ...
Creating namespace 'user-stage' ...
Creating certificates ...
Creating Kubectl credentials for 'user' ...
Creating Kubectl context 'user-context' for user 'user' ...
Creating role resources for user 'user' in namespace 'user-dev' ...
Creating role resources for user 'user' in namespace 'user-stage' ...
Switched to context "user-context".
Switching to namespace 'user-dev' ...
OK!
```

• Windows

In your command-line terminal, move to the D0100x-apps directory. Run the command `Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass`. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host D0100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
Bypass
```

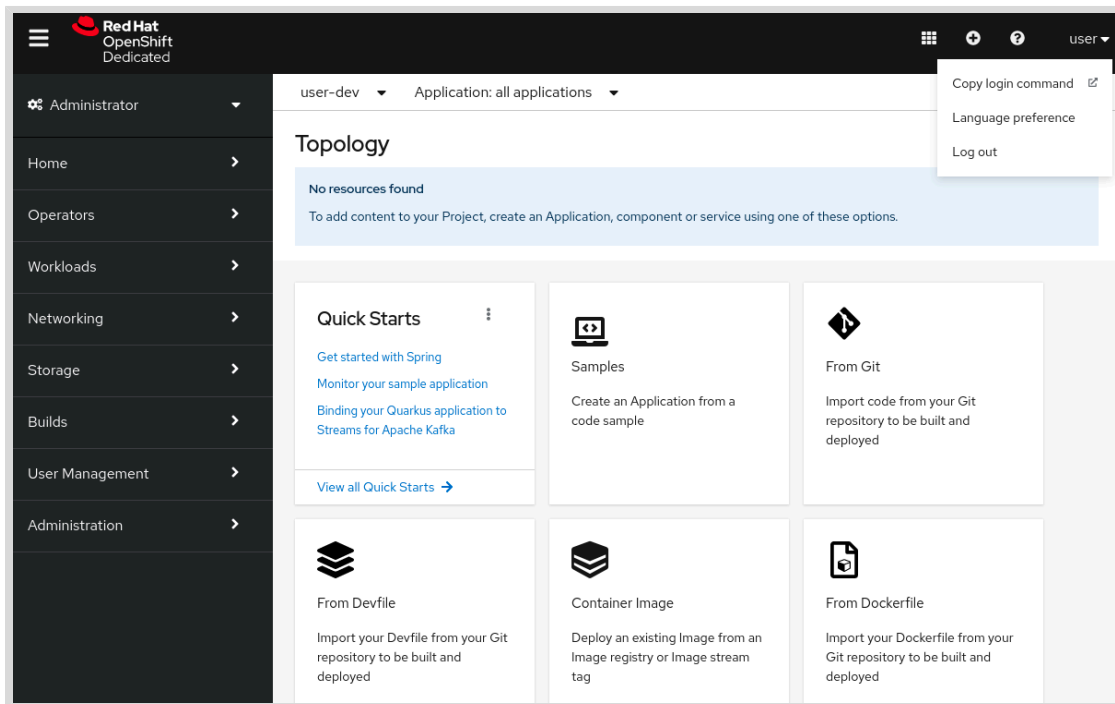
Run the script located at `.\setup\windows\setup.ps1`.

```
[user@host D0100x-apps]$ .\setup\windows\setup.ps1
Creating namespace 'user-dev' ...
Creating namespace 'user-stage' ...
Creating certificates ...
Creating Kubectl credentials for 'user' ...
Creating Kubectl context 'user-context' for user 'user' ...
Creating role resources for user 'user' in namespace 'user-dev' ...
Creating role resources for user 'user' in namespace 'user-stage' ...
Switched to context "user-context".
Switching to namespace 'user-dev' ...
OK!
```

5.2. Using the OpenShift Developer Sandbox

The `setup-sandbox` script will ask you to provide a server, a token and your username. To find the server and token follow these steps:

- Open a web browser and navigate to the OpenShift Developer Sandbox website. Log in with your username and password.
- Click on your username in the upper right pane of the screen. A dropdown menu opens.
- In the dropdown menu, click **Copy login command**. A new tab opens, log in again with your account if necessary by clicking **DevSanbox**.



- Click **Display Token**.
- The token you must provide in the script shows in your web browser.
- The server you must provide is a parameter of the `oc login` command displayed. For example, in the command `oc login --token=sha256~Gs54-tjq1Bo-fo866bddv8wbQ0bpmy321eSiqj1g --server=https://api.sandbox.x8i5.p1.openshiftapps.com:6443`, the server is `https://api.sandbox.x8i5.p1.openshiftapps.com:6443`.



- Keep these values. You will be asked for them in the script.

Run the appropriate script. The following instructions will depend on your operating system.

- **Linux and macOS**

In your command-line terminal, move to the `D0100x-apps` directory and run the script located at `./setup/operating-system/setup-sandbox.sh`. Replace

`operating-system` for `linux` if you are using Linux. Use `macos` if you are using macOS. Make sure the script has executable permissions.

```
[user@host D0100x-apps]$ chmod +x ./setup/operating-system/setup-sandbox.sh
[user@host D0100x-apps]$ ./setup/operating-system/setup-sandbox.sh
What is the OpenShift cluster URL?
https://api.sandbox.x8i5.p1.openshiftapps.com:6443
What is the OpenShift token?
sha256-wVSG...DDn0
What is your OpenShift username?
user
Creating Kubectl context...
Context created successfully
```

• Windows

In your command-line terminal, move to the `D0100x-apps` directory. Run the command `Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass`. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host D0100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
Bypass
```

Run the script located at `.\setup\windows\setup-sandbox.ps1`.

```
[user@host D0100x-apps]$ .\setup\windows\setup-sandbox.ps1
What is the OpenShift cluster URL?
https://api.sandbox.x8i5.p1.openshiftapps.com:6443
What is the OpenShift token?
sha256-wVSG...DDn0
What is your OpenShift username?
user
Creating Kubectl context...
Context created successfully
```

Finish

This concludes the guided exercise.

Running and Interacting with Your First Application

Objectives

After completing this section, you should be able to execute a pre-built application in your Kubernetes cluster and review the resources related to the process.

Running Pods From Container Images

The simplest way to run a container in your Kubernetes cluster is with the `kubectl run` command. At a minimum, you must specify a name and container image. This container image must be accessible by the Kubernetes cluster.

The following example command creates a new pod named `myname` that uses the container image referenced by `myimage`.

```
[user@host ~] kubectl run myname --image=myimage
```

Recent versions of `kubectl run` can only create new pods. For example, older example uses of this command might include a `--replicas` option, which has been removed.



Important

Use `kubectl run` to create pods for quick tests and experimentation.

Creating Resources

The `kubectl create` command creates new resources within the Kubernetes cluster. You must specify the name and type of the resource, along with any information required for that resource type.

You can specify `--dry-run=client` to prevent the creation of the object within the cluster. By combining this with the output type option, you can generate resource definitions.

For example, the following command outputs the YAML definition of a new deployment resource named `webserver`, by using the `nginx` image.

```
[user@host ~] kubectl create deployment webserver \
--image=nginx --dry-run=client -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: webserver
  name: webserver
spec:
  replicas: 1
  selector:
```

```

matchLabels:
  app: webserver
strategy: {}
template:
  metadata:
    creationTimestamp: null
    labels:
      app: webserver
  spec:
    containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}

```

You can save this output to a file to actually create the object later. Reference the file by specifying it with the `-f` option.

For example, the following command creates a new resource using the definition found in `mydef.yml`:

```

[user@host ~] kubectl create -f mydef.yml
...output omitted...

```

Comparing Create and Apply

The `kubectl create` command can only create *new* resources. To modify an existing resource or create it if it does not exist, use the `kubectl apply` command. Like `kubectl create`, this command also accepts YAML or JSON definitions.

If you are familiar with certain variants of SQL syntax, then `kubectl create` is comparable to `INSERT` whereas `kubectl apply` is akin to `UPSERT`.

Executing Commands Within an Existing Pod

With the `kubectl exec` command, you can execute commands inside *existing* pods. The `kubectl exec` command is useful for troubleshooting problematic containers, but the changes are not persistent. To make persistent changes, you must change the container image.

At a minimum, this command requires the name of the pod and the command to execute. For example, the following command will execute `ls` within the running pod named `mypod`.

```

[user@host ~] kubectl exec mypod -- /bin/ls
bin
boot
dev
...output omitted...

```

The `--` separates the parts of the command intended for Kubernetes itself from the command that should be passed to and executed within the container.

Connecting a Shell to an Existing Pod

A common use of `kubectl exec` is to open a new shell within a running pod. For example, the following command creates and attaches a new shell session to the pod named `mypod`:

```
[user@host ~] kubectl exec --stdin --tty mypod -- /bin/bash
```

Notice the addition of the `--stdin` and `--tty` options. These are necessary to ensure input and output are forwarded correctly to the interactive shell within the container.



References

Remove `kubectl run` generators PR

<https://github.com/kubernetes/kubernetes/pull/87077>

Get a Shell to a Running Container|Kubernetes

<https://kubernetes.io/docs/tasks/debug-application-cluster/get-shell-running-container/>

`kubectl` Cheat Sheet

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

► Guided Exercise

Running and Interacting with Your First Application

In this exercise, you will create a pod and connect to it. You will also create and manage a new namespace resource by using a resource definition file that you create.

Outcomes

You should be able to:

- Connect a shell session to an existing pod.
- Create a resource definition file.
- Use a resource definition to create and update a namespace resource.



Note

You do not need to understand Kubernetes namespaces to do this exercise, as they are solely used as an example resource.

Before You Begin

You need a working Kubernetes cluster, and your `kubectl` command must be configured to communicate with the cluster.

Ensure your `kubectl` context refers to the `user-dev` namespace. Use the `kubectl config set-context --current --namespace=user-dev` command to switch to the appropriate namespace.

Instructions

- 1. Use `kubectl run` and `kubectl exec` to create a new pod and attach a shell session to it.
 - 1.1. Create a new pod named `webserver` that uses the `httpd` container image.



Note

This course uses the backslash character (`\`) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (```) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host ~] kubectl run webserver \
--image=registry.access.redhat.com/ubi8/httpd-24:1-161
pod/webserver created
```

1.2. Confirm the `webserver` pod is running.

```
[user@host ~] kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
webserver	1/1	Running	0	5s

1.3. Connect to the pod by using `kubectl exec`.

```
[user@host ~] kubectl exec --stdin --tty \
webserver -- /bin/bash
root@webserver:/#
```

1.4. View the contents of the `httpd` configuration file within the pod.

```
[root@webserver:/#] cat /etc/httpd/conf/httpd.conf
...output omitted...
ServerAdmin root@localhost
...output omitted...
```

1.5. Disconnect from the pod by using the `exit` command.

```
[root@webserver:/#] exit
```

► **2.** Create a pod resource definition file and use that file to create another pod in your cluster.

2.1. Create a new file named `probes-pod.yml`. Add the following resource manifest to the file.

```
apiVersion: v1
kind: Pod
metadata:
  name: probes
  labels:
    app: probes
  namespace: user-dev
spec:
  containers:
  - name: probes
    image: 'quay.io/redhattraining/do100-probes:external'
    ports:
    - containerPort: 8080
```

Replace `user` by your OpenShift Developer Sandbox username.

You can generate the basic YAML file with the `kubectl run` command

```
[user@host ~] kubectl run probes
--image=quay.io/redhattraining/do100-probes:external --dry-run=client -o yaml
```

- 2.2. Use the `kubectl create` command to create a new pod from the resource manifest file.

```
[user@host ~] kubectl create -f probes-pod.yaml
pod/probes created
```

- 2.3. Verify that the pod is running.

```
[user@host ~] kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
probes	1/1	Running	0	15s
webserver	1/1	Running	0	10m

- ▶ 3. Modify the `metadata.labels.app` field of the pod manifest and apply the changes.

- 3.1. Open the `probes-pod.yaml` file. Replace `probes` by `do100-probes` in the `metadata.labels.app` field.

```
apiVersion: v1
kind: Pod
metadata:
  name: probes
  labels:
    app: do100-probes
...output omitted...
```

- 3.2. Attempt to update the pod by using the `kubectl create` command.

```
[user@host ~] kubectl create -f probes-pod.yaml
Error from server (AlreadyExists): error when creating "probes-pod.yaml": pods
"probes" already exists
```

Notice the error. Because you have previously created the pod, you can not use `kubectl create`.

- 3.3. Update the pod by using the `kubectl apply` command.

```
[user@host ~] kubectl apply -f probes-pod.yaml
pod/probes configured
```

**Note**

The preceding usage of `kubectl apply` produces a warning that the `kubectl.kubernetes.io/last-applied-configuration` annotation is missing. In most scenarios, this can be safely ignored.

Ideally, to use `kubectl apply` in this precise manner, you should use the `--save-config` option with `kubectl create`.

3.4. Verify that the label has been updated by using the `kubectl describe pod`.

```
[user@host ~] kubectl describe pod probes
Name:                probes
Namespace:            user-dev
...output omitted...
Labels:               app=do100-probes
...output omitted...
```

Finish

Delete the pod and namespace to clean your cluster.

```
[user@host ~]$ kubectl delete pod/webserver
pod "webserver" deleted
[user@host ~]$ kubectl delete pod/probes
pod "probes" deleted
```

This concludes the guided exercise.

Summary

In this chapter, you learned:

- Different application and organizational needs determine how you should run and configure your Kubernetes cluster.
- Kubernetes includes extensions to provide additional cluster functionality, such as storage and ingress.
- The command-line tool `kubectl` is the main way to interact and configure a Kubernetes cluster.